

Betriebssystemtechnik

Adressräume: Trennung, Zugriff, Schutz

V. Adressraumverwaltung

Wolfgang Schröder-Preikschat

12. Mai 2015



Linearer (eindimensionaler) Adressraum

Seitennummerierung steht für eine Unterteilung des Adressraums in gleichgroße Einheiten und deren lineare Aufzählung

- je nach Adressraumtyp werden diese Einheiten verschieden benannt

Seite (*page*) im logischen/virtuellen Adressraum

Seitenrahmen (*page frame*), auch Kachel, im realen Adressraum¹

- die vom Prozess generierte lineare Adresse la ist ein Tupel (p, o) :

- p ist eine Seitennummer (*page number*) im Adressraum $[0, 2^N - 1]$
 - Wertebereich für $p = [0, (2^N \text{ div } 2^O) - 1]$
- o ist der Versatz (*offset, displacement*) innerhalb von Seite p
 - Wertebereich für $o = [0, 2^O - 1]$

↔ mit $O \ll N$ und 2^O auch Seitengröße (in Bytes): typisch ist $2^{12} = 4096$

- tabellengesteuerte Abbildung von la mit p als Seitenindex

¹Hilfe: Die Bindung zwischen der Seite eines logischen Adressraums und einer Kachel ist eher fest – Seitenumlagerung (*swapping*) ist selten –, die zwischen der Seite eines virtuellen Adressraums und eines Seitenrahmens ist eher lose.



Gliederung

Seitennummerierung

Allgemeines
Abbildung

Segmentierung

Allgemeines
Abbildung

Übersetzungspuffer

Prinzip
Spülungssteuerung

Zusammenfassung



Seitendeskriptor I

Eintrag der Seiten-Kachel-Tabelle

Seitennummer bzw. Seitenindex identifizieren die die Adressabbildung steuernde und von der Hardware (MMU) vorgegebene Datenstruktur

- typischerweise umfassen die darin gebündelten Informationen:

Kachel-/Seitenrahmennummer ■ seitenausgerichtete reale Adresse
Attribute ■ Schreibschutzbit
■ Präsenzbit
■ Referenzbit²
■ Modifikationsbit²

- je nach Hardware und Adressraummodell gibt es weitere Attribute

- Privilegstufe, Seiten(rahmen)größe, Spülungssteuerung (TLB), ...

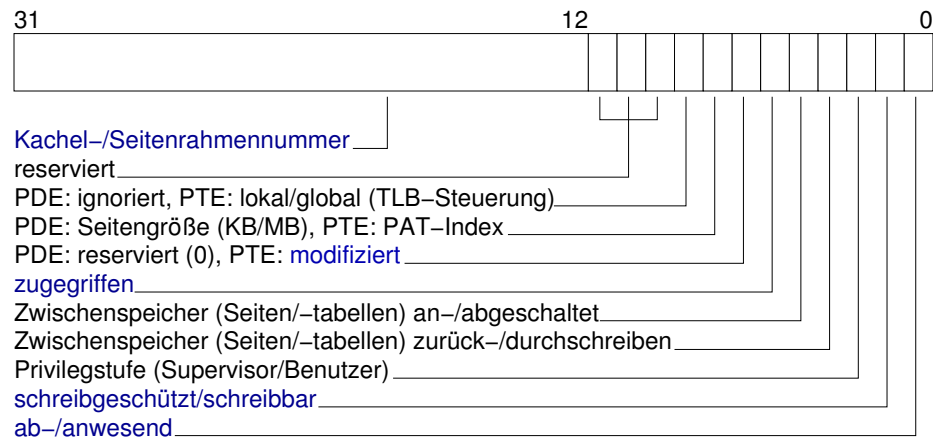
Betriebssysteme definieren pro Seitendeskriptor oft weitere Attribute, die im Schatten der Seiten-Kachel-Tabelle gehalten werden müssen

- Seitendeskriptor des Betriebssystems in der „*shadow page table*“

↔ die Struktur des Seitendeskriptors der Hardware ist unveränderlich

²„klebriges“ (*sticky*) Bit: wird von Hardware gesetzt aber nicht gelöscht.





PDE *page directory entry* PTE *page table entry* PAT *page attribute table*

```

1 struct ia32pd {
2     unsigned pd_present:1;
3     unsigned pd_writeable:1;
4     unsigned pd_supervisor:1;
5     unsigned pd_through:1;
6     unsigned pd_uncached:1;
7     unsigned pd_referenced:1;
8     unsigned pd_modified:1;
9     unsigned pd_index:1;
10    unsigned pd_global:1;
11    unsigned pd_avail:3;
12    unsigned pd_frame:20;
13 }; /* PTE */

1 struct ia32pd     pagetable     [PTECOUNT];
2 struct ia32pd_shadow pagetable_shadow[PTECOUNT];
    
```

- Ersetzungsstrategie [6]:
 - FIFO ■ Einlagerungszeit
 - Zeitstempel
 - LFU ■ Zugriffshäufigkeit
 - Zähler
 - LRU ■ *second chance* ...

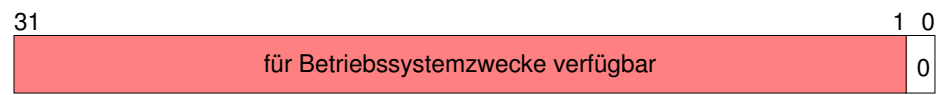
■ braucht Zusatzattribute

↪ **beachte:** ein Tabellenpaar pro Adressraum bzw. Prozessinkarnation

Auslagerung einer Seite bringt eine neue **Verortung** mit sich, über die Buch geführt werden muss

- ein weiterer möglicher Fall für einen Schattendeskriptor *oder*
 - wenn die Ortsinformation keinen Platz im realen Seitendeskriptor findet
 - weil sie zu groß ist oder der Deskriptoraufbau eine Aufnahme erschwert
 - Adresse im Hintergrundspeicher – ggf. sogar entfernter Hauptspeicher

■ der reale Seitendeskriptor ist passend auch dafür ausgelegt, IA-32:

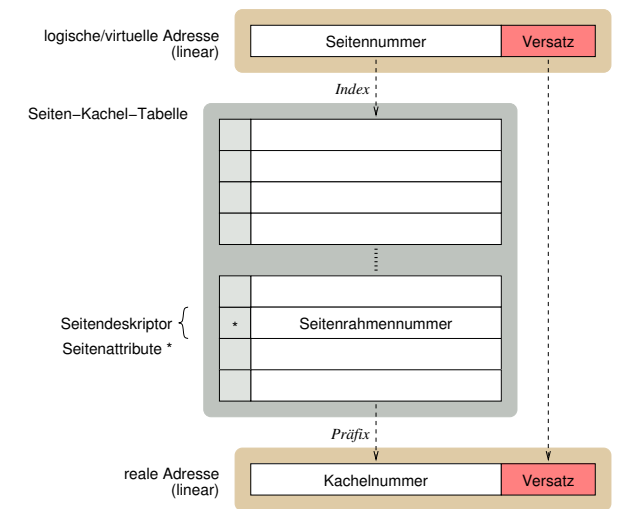


■ ist das Präsenzbit 0, werden Bits [1..31] nicht von der MMU genutzt

↪ bei Wiedereinlagerung sind „logisch invariante“ Attribute zu beachten

- dies betrifft Bits [1..11], deren Werte bei Auslagerung ggf. zu sichern sind
 - genauer: Bits [1..4] und [7..11]; Bits [5..6] sind „klebrige“ Bits³

³Bits [10..11] mit Bits [5..6] getauscht wäre „betriebssystemfreundlicher“.



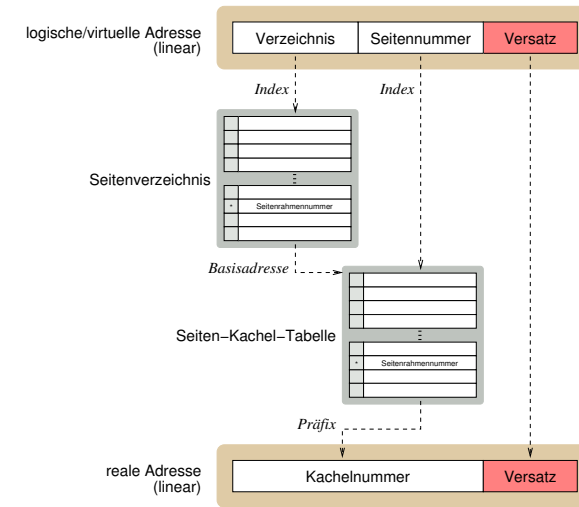
$$ra = (SKT[la / PSIZE].pd_frame * PSIZE) | (la \% PSIZE)$$

Adressbreite des Prozessors und Seitengröße⁴ haben großen Einfluss auf den Umfang der Seiten-Kachel-Tabelle

- Annahme: 32-Bit Maschine, 4 KiB Seite, 32-Bit Seitendeskriptor
 - Tabellengröße: $2^{32}/2^{12} = 2^{20}$ Einträge, einer pro Seite/Seitendeskriptor
 - Speicherplatzbedarf **pro logischen/virtuellen Adressraum**:
 - Seitendeskriptor: $32/8 = 2^2 = 4$ Bytes (je 4 KiB Seiten, wie z.B. bei IA-32)
 - Tabelle: $2^{20} * 2^2 = 2^{22} = 4$ MiB
 - genauer: **pro Prozessinkarnation** (Linux, MacOSX, Windows)
- Systemschnappschuss: MB Air 11", 4 GiB DDR, OSX 10.8.3
 - 92 Prozesse \rightsquigarrow 368 MiB \rightsquigarrow würden ca. 9% des Hauptspeichers sein !
- **beachte**: 64-Bit Maschine/Seitendeskriptor, 4 KiB Seite (z.B. IA-64)
 - $2^{64}/2^{12} = 2^{52} * 2^3 = 2^{55}$ Bytes pro Tabelle \approx **Petabytes** !!!
 - für große Adressräume ist die einstufige Abbildung **unpraktikabel** [4, 2]

↳ **Folge**: mehrstufige (2–5), invertierte oder segmentierte Tabellen

⁴Ist je nach Hardware (eingeschränkt) einstellbar durch das Betriebssystem.



```

1 SKT = SVZ[ 1a / (PTECOUNT * PSIZE)].pd_frame * PSIZE
2 ra = (SKT[(1a % (PTECOUNT * PSIZE)) / PSIZE].pd_frame * PSIZE) | (1a % PSIZE)
    
```

Diskussion II

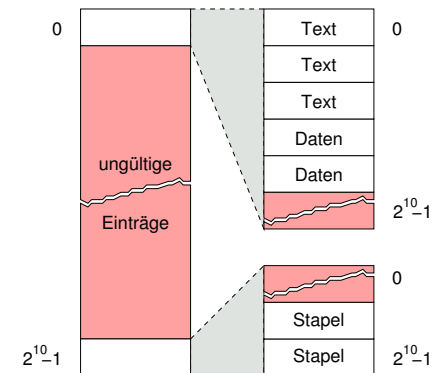
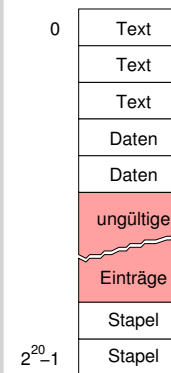
- Annahme wie auf S. 9, aber 10-Bit Verzeichnis- und Seitennummer
 - Größe der Verzeichnistabelle: $2^{10} = 1024$ Einträge je 4 Bytes \rightsquigarrow 4 KiB
 - Größe der Seiten-Kachel-Tabelle: *dito* \rightsquigarrow 4 KiB
- ↳ $2 * 4 = 8$ KiB pro $2^{10} * 2^{12} = 2^{22} = 4$ MiB Adressraumabschnitt
- $1 < n \leq 2^{10}$ Seitentabellen pro Adressraum bzw. Prozessinkarnation
 - die Tabellenanzahl wird von verschiedenen Faktoren beeinflusst:
 - getrennte Abbildung verschiedener Text-/Datenbereiche eines Programms
 - Einblendung des aktuellen Benutzeradressraums in den Kernadressraum
 - Zugriff auf gemeinsame Bibliotheken oder Speicherbereiche (inkl. E/A)⁵
 - nicht zuletzt: die statische/dynamische Größe des abzubildenden Programms
- pro (UNIX-) Prozess ergeben sich wenigstens drei Deskriptortabellen:
 - i eine Seitentabelle für die Text- und Datenbereiche des Prozesses
 - ii eine Seitentabelle für den Stapelspeicher des Prozesses
 - iii eine Verzeichnistabelle mit je einem Eintrag für diese Seitentabellen

↳ **beachte**: 64-Bit Adressen implizieren eine bis zu 5-stufige Abbildung

⁵speicherabgebildete Ein-/Ausgabekanäle (*memory mapped I/O*)

Tabellenstruktur eines Prozessadressraums

- ein Prozess belege 12 KiB Text, 8 KiB Daten und 8 KiB Stapel
 - Annahmen über die Hardwareorganisation wie auf S. 9 und 11
- einstufige Tabelle
- zweistufige Tabelle



- $2^{20} - 7$ ungültige Einträge
 - auf 1 Tabelle
- 3063 ungültige Einträge
 - auf 3 Tabellen

Invertierte Seitentabelle

Konzept einer Abbildungstabelle für den Hauptspeicher, d.h., dem „speicherbestückten realen Adressraum“⁶

- pro Kachel/Seitenrahmen gibt es einen Deskriptor, nicht pro Seite
 - die Tabelle reflektiert den realen Adressraum, nicht logischen/virtuellen
 - abgebildet wird Kachel-/Seitenrahmennummer auf Tupel (aid, p)
 - aid ist die **Adressraumidentifikation** (eines geladenen Programms)
 - p ist eine **Seitennummer** (vgl. S. 3) im Adressraum aid
- ↪ der zugehörige Tabellenindex ist die Kachel-/Seitenrahmennummer
- anstatt indizierte Adressierung mit p erfolgt die **Suche** nach (aid, p)
 - kritischer Faktor dabei ist die gegebene **Kachel-/Seitenrahmennzahl**:
 - i Assoziativregister bzw. -speicher für die Seiten-Kachel-Tabelle und/oder
 - ii Streuwertfunktion mit Streuwertankertabelle (*hash anchor table*), sowie Kollisionserkennung und -behandlung (Seitendeskriptorverkettung)
 - gestreut invertierte Seitentabellen eingeführt mit IBM System/38 [4]

⁶inkl. speicherabgebildete Ein-/Ausgabe (*memory mapped I/O*)

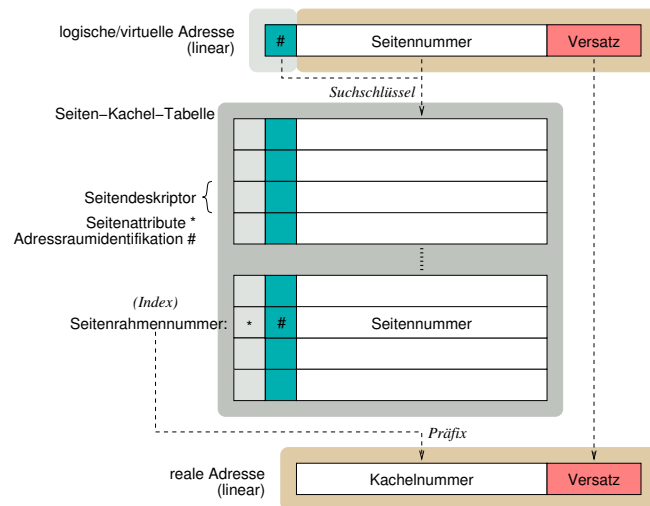
Linear invertierte Seitentabelle

- eine **zentrale Tabelle** für jeden Prozessor(kern) im System
 - alle Prozessinkarnationen teilen sich dieselbe Seiten-/Kacheltabelle
 - aid ist eine Prozessidentifikation, um gleiche p zu unterscheiden
 - Dimensionierung nach max. Kachel-/Seitenrahmennzahl ist kritisch:⁷
 - i 32-Bit Adressbus: $2^{32}/2^{12} = 2^{20}$ Einträge je 6 B \sim 6 MiB
 - ii 44-Bit Adressbus: $2^{44}/2^{12} = 2^{32}$ Einträge je 6 B \sim 24 GiB
 - iii 50-Bit Adressbus: $2^{50}/2^{12} = 2^{48}$ Einträge je 6 B \sim 281 TiB
 - größere Seiten und/oder Größe des Hauptspeichers zu Grunde legen
 - d.h., alle speicherabgebildeten adressierbaren Entitäten der Hardware
 - Problem dabei sind Lücken im realen Adressraum zwischen den Entitäten
- ↪ **beachte**: „Speicher“ im realen Adressraum muss nicht linear angeordnet sein
- eine **individuelle Tabelle** für jede Prozessinkarnation \sim variabel
 - Prozesswechsel bedingt **Tabellenumschaltung** – ggf. TLB-Spülung (S. 28)
 - aid ist der Zeiger auf die Seiten-Kachel-Tabelle der Prozessinkarnation
 - die Tabellengröße richtet sich nach der Größe des Prozessadressraums

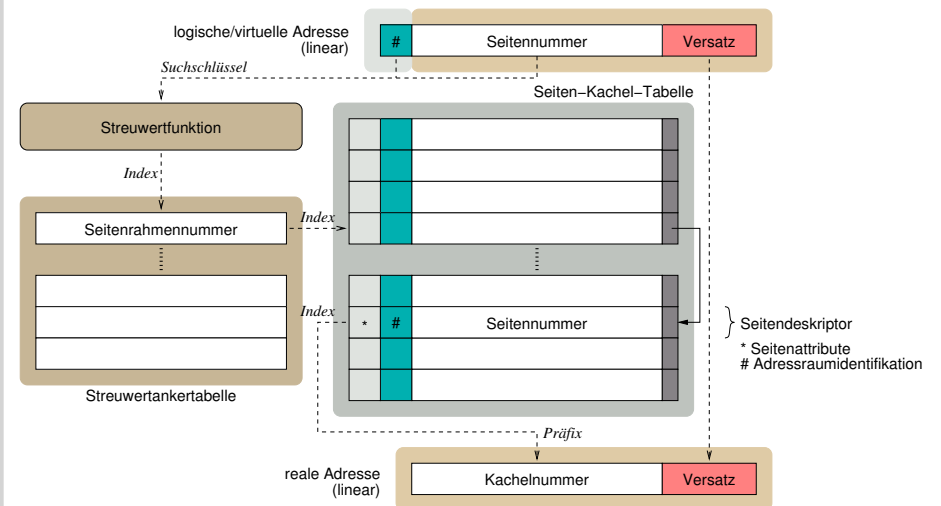
⁷Jeder Eintrag für $(aid, p) \mapsto 2 + 4$ Bytes angenommen.

Seitenbasierte Adressierung: linear invertiert

zentral



Seitenbasierte Adressierung: gestreut invertiert



Zielkonflikt zwischen Speicherbedarfs- und Leistungsminimierung zur tabellengesteuerten Umsetzung des Abbildungsprozesses

- speicherschonende und rechenintensive Repräsentation

Tabellenorganisation	Betriebssystem	Prozessor (MMU)
mehrstufig	komplex	komplex
gestreut invertiert	einfach	komplex
variabel linear invertiert	einfach	komplex

- speicherintensive und bedingt rechenschonende Repräsentation

Tabellenorganisation	Betriebssystem	Prozessor (MMU)
einstufig	einfach	einfach
linear invertiert	einfach	komplex



Seitennummerierung
Allgemeines
Abbildung

Segmentierung
Allgemeines
Abbildung

Übersetzungspuffer
Prinzip
Spülungssteuerung

Zusammenfassung



Nichtlinearer (zweidimensionaler) Adressraum

Segmentierung steht für eine Strukturierung des Adressraums in Einheiten von möglicherweise verschiedener Größe

- diese Einheiten werden als **Segment** bezeichnet
 - sie setzen sich zusammen aus gleichgroßen Elementen und
 - bilden eine lineare Folge von „Granulaten“ (Bytes, Seiten):
 - Byte \mapsto unbedingt zusammenhängend auch im realen Adressraum
 - Seite \mapsto bedingt: **seitennummerierte Segmentierung** (*paged segmentation*)
- die vom Prozess generierte Adresse la bildet ein Paar (S, A) :
 - S ist **Segmentname** (auch Segmentnummer) \rightsquigarrow 1. Dimension
 - Wertebereich für $S = [0, 2^M - 1]$; bei IA-32: $M = 13$
 - A ist **Adresse**, auch **Versatz**, innerhalb des Segments S \rightsquigarrow 2. Dimension
 - Wertebereich für $A = [0, 2^N - 1]$
- tabellengesteuerte Abbildung von la mit S als **Segmentindex**
 - zur Auswahl des für S gültigen Segmentdeskriptors in der Segmenttabelle



Segmentdeskriptor I

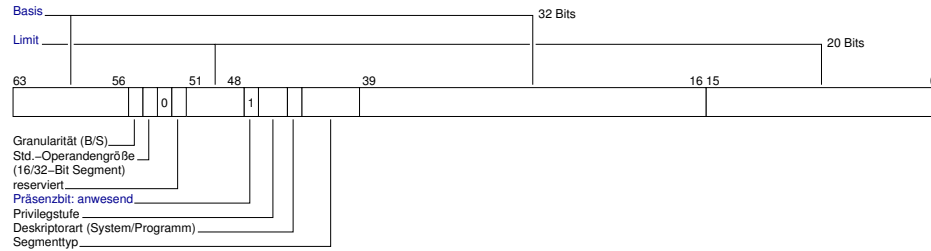
Eintrag der Segmenttabelle

Segmentname/-index identifizieren die die Adressabbildung steuernde und von der Hardware (MMU) vorgegebene Datenstruktur

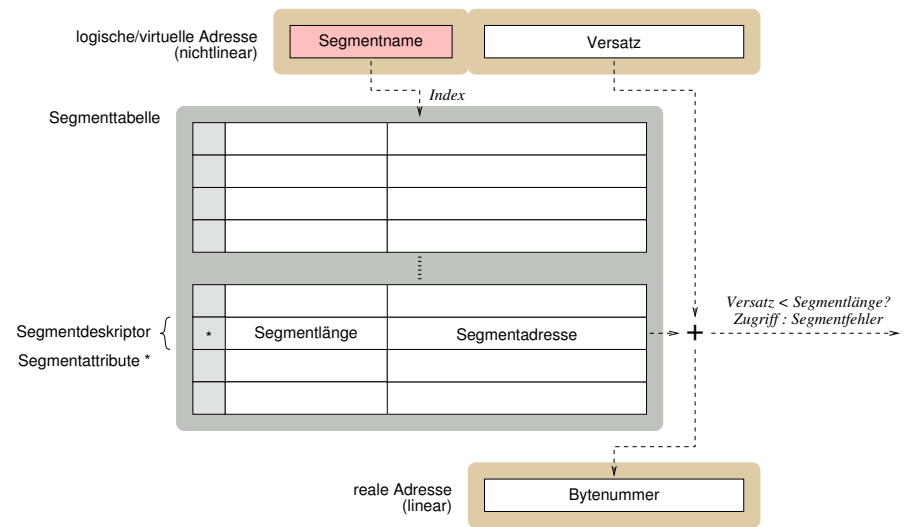
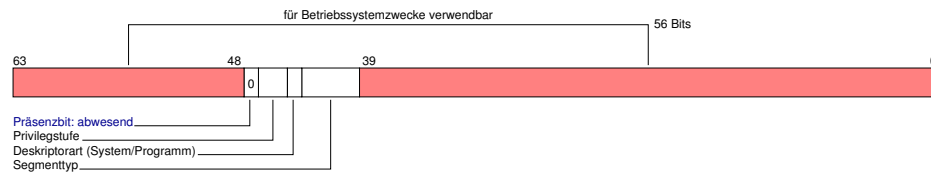
- typischerweise umfassen die darin gebündelten Informationen:
 - Basis**
 - Segmentanfangsadresse im Arbeitsspeicher
 - Ausrichtung (*alignment*) entsprechend der Granulatgröße
 - Limit**
 - Segmentlänge als Anzahl der Granulate
 - Zahl der aufeinanderfolgenden Granulatadressen
 - Attribute**
 - Typ (Text, Daten, Stapel)
 - Zugriffsrechte (lesen, schreiben, ausführen)
 - Expansionsrichtung (auf-/abwärts)
 - Präsenzbit
- je nach Hardware und Adressraummodell gibt es weitere Attribute
 - Privilegstufe, Klasse (*interrupt, trap, task*), Granulatgröße, ...
 - Seiten-Kachel-Tabelle (seitennummerierte Segmentierung)



ein als „anwesend“ markiertes Segment

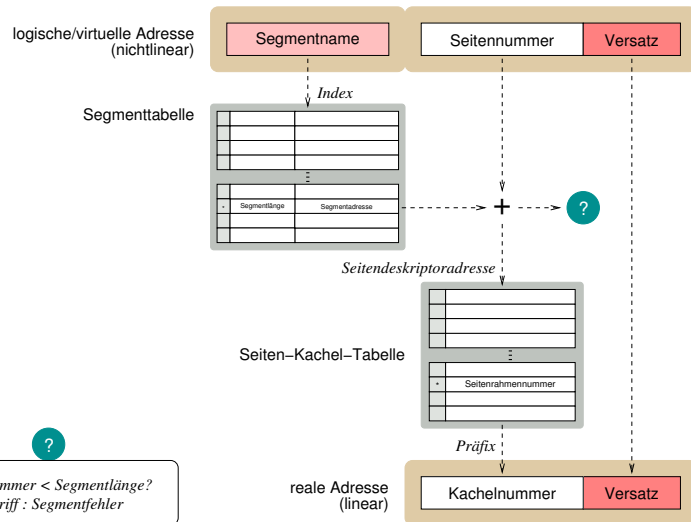


ein als „abwesend“ markiertes Segment



$$1 \quad ra = la < ST[sn].sd_limit ? ST[sn].sd_base + la : segmentation\ violation \rightsquigarrow trap$$

Segmentbasierte Adressierung: seitennummeriert



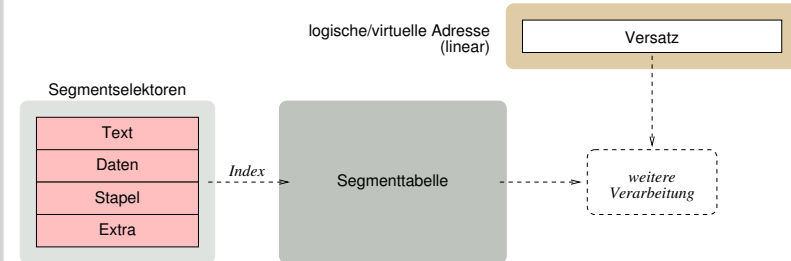
$$1 \quad SKT = (la / PSIZE) < ST[sn].sd_limit ? ST[sn].sd_base : segmentation\ violation \rightsquigarrow trap$$

$$2 \quad ra = (SKT[la / PSIZE].pd_frame * PSIZE) | (la \% PSIZE)$$

Segmentbasierte Adressierung: implizit

Segmentregister bzw. Segmentselektor (segment selector)

- je nach Zugriffsart selektiert die MMU implizit das passende Segment



- Befehlsabruf (*instruction fetch*) aus Textsegment
 - Operantionscode \mapsto Segmentname „Text“
- Operandenabruf (*operand fetch*) aus Text-, Daten-, Stapelsegment
 - Direktwerte \mapsto Segmentname „Text“
 - globale/lokale Daten \mapsto Segmentname „Daten“ \equiv „Stapel“

- Programme können weiterhin eindimensionale Adressen verwenden

Seitennummerierung

Allgemeines
Abbildung

Segmentierung

Allgemeines
Abbildung

Übersetzungspuffer

Prinzip
Spülungssteuerung

Zusammenfassung



Zwischenspeicherung des Ergebnisses des Übersetzungsprozesses in einen als **Assoziativspeicher** organisierter Puffer

- erstmalig umgesetzt in IBM System/370 [5, 1]
 - segmentbasierte seitennummerierte Adressierung (31-Bit Format)⁸
 - Segment- und Seitenindex (der virtuellen Adresse) als Suchschlüssel
 - direkte Abbildung auf die Kachelnummer bei einem Treffer
 - 8 – 128 Puffereinträge, je nach Systemkonfiguration
 - Einträge sind (Unter- oder Obermengen von) Seitendeskriptoren
- Umsetzungsfehler** (*lookup miss*) ziehen eine Wanderung über ein oder mehrere Tabellen (*table walk*) nach sich
- beim hardware-geführten TLB läuft die CPU/MMU die Tabellen ab
 - der Seitenfehler kommt verzögert, bei erfolgloser Tabellenwanderung
 - beim software-geführten TLB ist dies eine Betriebssystemfunktion [7]
 - der Seitenfehler kommt unverzögert und unbedingt: RISC-Ansatz

⁸Ist das höchstwertige Bit (Bit₀) einer 32-Bit Adresse 1, handelt es sich um eine 31-Bit virtuelle Adresse. Ansonsten um eine 24-Bit Adresse.



Zwischenspeicher von Adressabbildungen

Übersetzungspuffer unterscheiden sich von Zwischenspeichern für Programmtext und -daten

- zwischengespeichert werden (reale) Adressen, nicht deren Inhalte
- genauer: es werden Inhalte von (Seiten-) Deskriptoren gepuffert

Kontextwechsel (zwischen Prozessen/Adressräumen) implizieren Maßnahmen zur **Vereindeutigung** gepufferter Einträge

- ausspülen (*flush*) des gesamten TLB
 - unbedingte Folge sind Zugriffsfehler durch den angeschalteten Prozess
 - ist durchaus praktikabel bei einem vergleichsweise kleinen TLB
- beschildern (*tag*) einzelner Einträge im TLB
 - bei zu kleinem TLB droht häufiger Überlauf und Leistungseinbuße
 - bevorzugt verdrängt werden „falsch beschilderte“ Einträge
 - die nicht dem gegenwärtigen Prozess entstammen

↪ ein software-geführter TLB bietet viele Optionen und Flexibilität



Beschilderung der Puffereinträge

Tagged TLB [3]

- Adressraumbezeichner (*address-space identifier*, ASID)
 - identifiziert die einem TLB-Eintrag zugehörige Prozessinkarnation
 - Seitennummer und ASID-Register⁹ bilden den Suchschlüssel
 - z.B. Alpha, MIPS, (mancher) PowerPC und UltraSPARC
- Bereichsbezeichner (*region identifier*, RID)
 - Generalisierung des ASID-Schemas: mehrere RID können aktiv sein
 - führende Adressbits (*virtual region number*) selektieren Bereichsregister
 - z.B. IA-64 und PA-RISC
- Schutzschlüssel (*protection key*, PK)
 - ASID-Alternative: dienen nicht direkt der Suche nach einem TLB-Eintrag
 - assoziative Suche nach Schutzschlüsselregister (*protection key register*)⁹
 - z.B. IA-64 und PA-RISC
- Domänenbezeichner (*domain identifier*, DID)
 - Schutzschlüsseln sehr ähnlich, liefert jedoch viel weniger Beschilderungen
 - keine assoziative Suche, stattdessen Indizierung eines Domänenregisters
 - z.B. ARM

⁹Inhalt ist Teil des Prozesskontextes.



Seitennummerierung

Allgemeines
Abbildung

Segmentierung

Allgemeines
Abbildung

Übersetzungspuffer

Prinzip
Spülungssteuerung

Zusammenfassung



■ Seitennummerierung

- linearer (eindimensionaler) Adressraum
- Seitendeskriptoren und -tabellen
- ein-/mehrstufige seitenbasierte Adressierung
- Tabellenstruktur eines Prozessadressraums
- linear/gestreut invertierte Seitentabelle und Adressierung

■ Segmentierung

- nichtlinearer (zweidimensionaler) Adressraum
- Segmentdeskriptoren und -tabellen
- segmentbasierte (seitennummerierte) Adressierung
- implizite Selektion von Segmentdeskriptoren, je nach Zugriffsart (IA-32)

■ Übersetzungspuffer

- hard- und softwaregeführter TLB
- Spülung bzw. Spülungssteuerung des TLB
- Beschilderung von Puffereinträgen: ASID, RID, PK, DID



Literaturverzeichnis I

- [1] CASE, R. P. ; PADEGS, A. :
Architecture of the IBM System/370.
In: *Communications of the ACM* 21 (1978), Jan., Nr. 1, S. 73–96
- [2] CHANG, A. ; MERGEN, M. F.:
801 Storage: Architecture and Programming.
In: *ACM Transactions on Computer Systems* 6 (1988), Febr., Nr. 1, S. 28–50
- [3] HEISER, G. :
Dealing with TLB Tags or I Want to Build a System, What Can L4 Do for Me?
In: ELPHINSTONE, K. (Hrsg.): *Proceedings of the 2nd Workshop on Microkernels and Microkernel-Based Systems*, 2001, S. 8
- [4] HOUDEK, M. E. ; MITCHELL, G. R.:
Translating a Large Virtual Address.
In: UTLEY, B. G. (Hrsg.): *IBM System/38 Technical Developments*.
IBM General Systems Division, Dez. 1978, S. 22–24
- [5] IBM CORPORATION (Hrsg.):
IBM System/370 Principles of Operation.
Fourth.
White Plains, NY, USA: IBM Corporation, Sept. 1 1975. –
GA22-7000-4, File No. S/370-01



Literaturverzeichnis II

- [6] SCHRÖDER-PREIKSCHAT, W. ; KLEINÖDER, J. :
Systemprogrammierung.
http://www4.informatik.uni-erlangen.de/Lehre/WS08/V_SP, 2008 ff.
- [7] UHLIG, R. ; NAGLE, D. ; STANLEY, T. ; MUDGE, T. ; SECHREST, S. ; BROWN, R. :
Design Tradeoffs for Software-Managed TLBs.
In: *ACM Transactions on Computer Systems* 12 (1994), Aug., Nr. 3, S. 175–205

